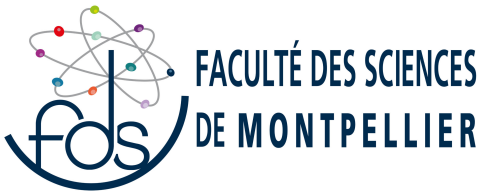


Lecture d'article

ADAM pour le Deep Learning

Groupe :

ACHIQ Aya
CLETZ Laura
EL MAZZOUJI Wahel

Octobre 2025

Table des matières

1 Introduction **2**

2 Adam et les méthodes adaptatives **2**

2.1 Contexte et motivation 2

2.2 Principe de l'algorithme Adam 2

2.3 Interprétation et propriétés 2

2.4 Forces et apports de la méthode Adam 2

2.5 Limites théoriques 3

3 Expériences et comparaisons **3**

3.1 Base de données "Credit Card Fraud Detection" 3

3.2 Base de données "" 5

4 Conclusion **5**

A Annexes **6**

1 Introduction

2 Adam et les méthodes adaptatives

2.1 Contexte et motivation

L'apprentissage profond repose sur la minimisation stochastique d'une fonction de coût bruitée. La descente de gradient stochastique et ses variantes à momentum sont simples mais sensibles au choix du taux d'apprentissage et instables en présence de gradients bruités ou clairsemés. Pour pallier ces limites, Adam (*Adaptive Moment Estimation*) a été proposé par **kingma2014**. L'idée est de combiner les avantages du momentum (cf annexes A) et de l'adaptation du taux d'apprentissage, comme dans AdaGrad et RMSProp (cf annexes A), afin d'obtenir une descente plus rapide et plus stable.

Cependant, plusieurs travaux ont montré que la rapidité de convergence des méthodes adaptatives ne garantit pas toujours une meilleure généralisation. **wilson2017** soulignent que ces méthodes peuvent conduire à des minima différents de ceux trouvés par SGD, parfois moins performants sur les données de test.

2.2 Principe de l'algorithme Adam

L'algorithme Adam combine deux idées majeures : l'accumulation du momentum et l'adaptation du taux d'apprentissage pour chaque paramètre. À chaque itération t, le gradient stochastique $g_t = \nabla_\theta f_t(\theta_{t-1})$ sert à mettre à jour deux moyennes mobiles : $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ et $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$, où m_t représente la moyenne et v_t la variance du gradient. Les coefficients usuels sont $\beta_1 = 0.9$ et $\beta_2 = 0.999$. Comme m_0 et v_0 sont initialisés à zéro, Kingma et Ba introduisent une correction de biais : $\hat{m}_t = m_t/(1-\beta_1^t)$ et $\hat{v}_t = v_t/(1-\beta_2^t)$.

La mise à jour s'écrit

$$\theta_t = \theta_{t-1} - \alpha \, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon},$$

où α est le taux d'apprentissage et $\varepsilon \approx 10^{-8}$ évite la division par zéro. Cette règle combine la direction moyenne du gradient (momentum) et une normalisation par la variance locale, assurant une descente rapide et stable face aux variations d'échelle.

2.3 Interprétation et propriétés

Adam combine momentum et adaptation du pas : m_t lisse la direction de descente, tandis que v_t ajuste le pas selon la variance locale du gradient, réduisant les oscillations et stabilisant l'entraînement.

Adam présente plusieurs propriétés remarquables. Il est invariant à l'échelle des gradients, son pas est borné et contrôlé par α, et il reste peu sensible aux hyperparamètres. Kingma et Ba montrent, dans le cadre convexe, un regret en $\mathcal{O}(\sqrt{T})$, équivalent à AdaGrad mais plus stable sur des objectifs non stationnaires.

Ces propriétés expliquent la popularité d'Adam pour l'apprentissage de réseaux profonds, où les gradients peuvent être bruités, corrélés ou fortement déséquilibrés selon les paramètres.

2.4 Forces et apports de la méthode Adam

Les expériences de **kingma2014** sur diverses tâches, telles que la régression logistique et l'apprentissage de réseaux profonds, montrent qu'Adam combine la rapidité d'AdaGrad et la

stabilité de RMSProp. Sur MNIST, il atteint une faible perte en peu d'itérations et une précision comparable à SGD avec momentum, sans réglage manuel du taux d'apprentissage.

L'algorithme se distingue par sa robustesse aux hyperparamètres, sa rapidité de convergence et son efficacité face à des gradients bruités. Les valeurs par défaut ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$) offrent de bonnes performances dans la plupart des contextes, ce qui explique sa popularité comme méthode d'optimisation de référence en apprentissage profond.

Enfin, la variante AdaMax, fondée sur la norme infinie, améliore la stabilité numérique tout en conservant la simplicité d'Adam.

2.5 Limites théoriques

Bien qu'efficace en pratique, Adam présente une généralisation limitée. **wilson2017** montrent que les méthodes adaptatives diffèrent des approches non adaptatives comme SGD.

Théoriquement, **wilson2017** montrent que SGD converge vers une *solution à norme minimale* ($\|w\|_2$ faible), gage d'une bonne généralisation, tandis que les méthodes adaptatives tendent vers des *solutions à norme infinie minimale* ($\|w\|_\infty$ faible), souvent associées à un surapprentissage.

wilson2017 montrent qu'Adam peut converger vers des solutions mal généralisées, alors que SGD atteint la solution optimale, ce qui explique la moindre généralisation des méthodes adaptatives.

Ces travaux soulignent ainsi le compromis fondamental entre rapidité d'apprentissage et capacité de généralisation propre aux méthodes adaptatives.

3 Expériences et comparaisons

Suivant les résultats théoriques comparatifs de **wilson2017**, nous allons comparer sur deux bases de données connues les méthodes Adam et SGD, mais aussi deux méthodes proches : Adagrad et RMSprop.

3.1 Base de données "Credit Card Fraud Detection"

Pour comparer les performances des quatre méthodes citées, nous commençons avec la base de données "Credit Card Fraud Detection". C'est une base de données publique disponible sur Kaggle [1] qui contient des transactions par carte de crédit, dont certaines sont frauduleuses. Le but est de détecter ces fraudes à partir des caractéristiques des transactions.

Nous gardons les variables "V14" et "Amount" comme variables explicatives, elles représentent respectivement les résultats d'une ACP (Analyse en Composantes Principales) anonymisant ainsi les données d'utilisation et d'utilisateur.ice de la carte qui sont fortement corrélés à la fraude, et le montant de la transaction. La variable "Class" est la variable réponse où 0 signifie que la transaction est légitime et 1 est une fraude.

Nous utilisons un réseau de neurones simple avec une couche cachée de 16 neurones et une fonction d'activation ReLU (cf cours GLM). La fonction de perte "`binary cross-entropy`" est en fait celle associée à la régression logistique (cf annexes A).

Nous entraînons le modèle avec les quatre méthodes d'optimisation : Adam, SGD, Adagrad et RMSprop, en utilisant les paramètres par défaut recommandés dans la littérature, à savoir un taux d'apprentissage $\eta = 0.001$.

1. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Nous trouvons pour des échantillons de taille $n = 10\,000$ et $n = 100\,000$ les résultats suivants :

Optimiseur	Test Loss	Test Accuracy	Erreur de test (%)
Adam	0.0048	0.9977	0.23
SGD	0.0047	0.9980	0.20
Adagrad	**0.0035**	0.9977	0.23
RMSprop	0.0071	**0.9980**	**0.20**

TABLE 1 – Comparaison des performances sur l'échantillon de taille $n = 10\,000$

Optimiseur	Test Loss	Test Accuracy	Erreur de test (%)
Adam	0.0034	**0.9991**	**0.09**
SGD	**0.0033**	**0.9991**	**0.09**
Adagrad	0.0109	0.9986	1.40
RMSprop	0.0059	0.9987	1.30

TABLE 2 – Comparaison des performances sur l'échantillon de taille $n = 100\,000$

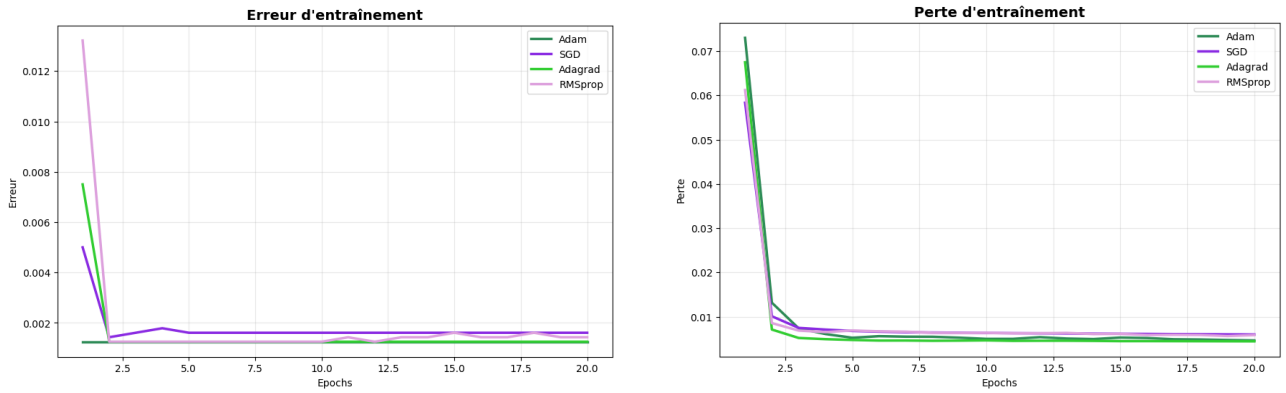
Les tableaux 1 et 2 révèlent plusieurs tendances intéressantes :

Sur un 10 000-échantillon, Adagrad obtient la meilleure perte de test (0.0035), tandis que SGD et RMSprop partagent la meilleure précision (99.80%). Les différences restent néanmoins marginales.

Sur un 100 000-échantillon, Adam et SGD dominent avec une précision remarquable de 99.91% et des erreurs de test identiques (0.09%). SGD présente légèrement la meilleure perte (0.0033). Adagrad et RMSprop montrent des performances dégradées sur ce plus grand échantillon.

Il semble ainsi que l'augmentation de la taille d'échantillon favorise nettement Adam et SGD, confirmant leur robustesse sur des datasets plus importants, tandis que les méthodes adaptatives classiques (Adagrad, RMSprop) semblent moins performantes avec plus de données.

Graphiquement, ces observations se confirment à travers l'évolution des métriques d'entraînement et de test. La figure 1 illustre l'évolution de l'erreur d'entraînement au cours des *epochs*, révélant les vitesses de convergence distinctes de chaque optimiseur :



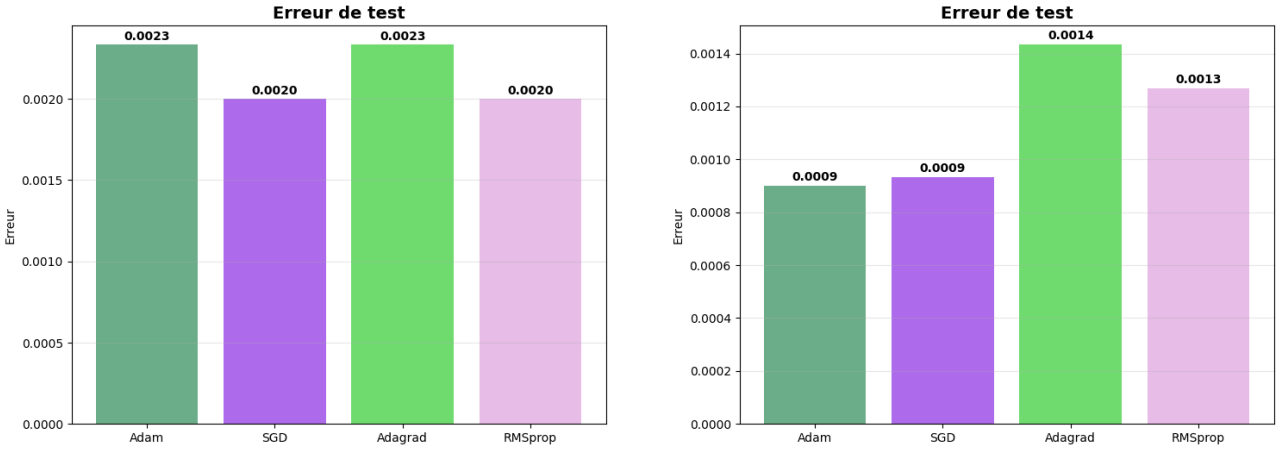
(a) Erreur d'entraînement (b) Perte d'entraînement

FIGURE 1 – Comparaison des métriques d'entraînement pour $n = 10\,000$

L'erreur d'entraînement est la proportion de prédictions incorrectes sur l'ensemble d'entraînement, soit 1− la précision durant l'entraînement (*training accuracy*), tandis que la perte mesure la qualité globale des prédictions.

On observe qu'Adagrad converge rapidement vers une faible erreur initiale, mais stagne ensuite, tandis qu'Adam et SGD continuent à améliorer l'erreur au fil des *epochs*. RMSprop montre une convergence plus lente mais régulière.

La figure 2 présente une synthèse des performances finales de test, mettant en évidence les différences de généralisation entre les méthodes selon la taille de l'échantillon :



(a) Erreurs de test ($n = 10\,000$)

(b) Erreurs de test ($n = 100\,000$)

FIGURE 2 – Comparaison des erreurs de test finales

Ces visualisations confirment que SGD et Adam maintiennent des performances robustes à travers les différentes tailles d'échantillon, tandis qu'Adagrad, malgré une convergence rapide sur le petit échantillon, voit ses performances se dégrader significativement avec l'augmentation des données.

3.2 Base de données ""

4 Conclusion

A Annexes

Gradient non adaptatif

Une méthode non adaptative utilise un taux d'apprentissage fixe η identique pour tous les paramètres et à toutes les itérations :

$$\theta_{t+1} = \theta_t - \eta\, g_t,$$

où $g_t = \nabla_\theta f_t(\theta_t)$ est le gradient de la fonction de coût.

Gradient adaptatif

Une méthode adaptative ajuste dynamiquement le taux d'apprentissage pour chaque paramètre en fonction de l'historique de ses gradients. Les paramètres dont les gradients varient fortement reçoivent un pas plus petit, ceux dont les gradients sont faibles un pas plus grand :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon}\, g_t,$$

avec v_t la moyenne des carrés des gradients passés.

Momentum

Le momentum ajoute une mémoire du gradient passé pour lisser la trajectoire de la descente :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t.$$

Ce mécanisme stabilise la descente et permet d'atteindre plus facilement le minimum.

Adagrad

Adagrad adapte le taux d'apprentissage en fonction de l'historique des gradients :

$$v_t = v_{t-1} + g_t^2,$$

où v_t est la somme des carrés des gradients. La mise à jour devient

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon}\, g_t.$$

RMSProp

RMSProp modifie Adagrad en utilisant une moyenne mobile exponentielle des carrés des gradients :

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

avec β_2 proche de 1. La mise à jour est la même.

Régression logistique

La régression logistique est un type de régression dans les cas binaires. Sa fonction de perte est définit par

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

avec $y \in 0, 1$ et $\hat{y} = \sigma(z)$ la probabilité prédite via la fonction sigmoid $\sigma(z) = 1/(1 + e^{-z})$.